Announcements

- Cross-check all scores on Ted
- Two more blog deadlines: Sunday and Wednesday
- Final Project Presentations in CSE 1202, Thursday 3-6pm
  - Bring computer with VGA adapter
    - contact instructor if this is not an option for you
- CAPE reminder
Lecture Overview

- Volume Rendering
  - Overview
  - Transfer Functions
  - Rendering
What is Volume Rendering?

- *Volume Rendering* is a set of techniques used to display a 2D projection of a volume data set.
- User specifies viewpoint, rendering algorithm and transfer function
  - Transfer function defines mapping of color and opacity to voxels
- Wide spectrum of application domains
Weather

- Clouds
  - http://www.youtube.com/watch?v=7obZdsEoGGA
Lecture Overview

- Volume Rendering
  - Overview
  - Transfer Functions
  - Rendering
Transfer Functions

- Most volume data sets consist of a single data value per voxel, for instance the measured material density.
- This data value can be interpreted as:
  - Luminance and rendered as a gray value on a scale from black to white
  - An index into a color look-up table
- Another look-up table maps opacity to data values.

From GPU Gems
Opacity Transfer Function

- Modifying the mapping of data value to opacity exposes different parts of the volume.

From Shin et al. 2004: Images a-f show decreasing opacity.
Volume Filtering

- Applying a filter to the volume data set can improve image quality
- Filtering operation defined by filter kernel
- Filter kernels on right:
  - (a) blur filter
  - (b) sharpen filter
  - (c) edge filter
- In 3D, filter kernels typically use a 6-, 18- or 26-voxel neighborhood
Derived Voxel Data

- Applying a filter on the volume data set generates a new, derived volume data set.
- Derived volume data can be stored with original volume in a multi-channel volume data set.

Example:
- Channel 1: original density data
- Channel 2: gradient magnitude
2D Transfer Functions

- A 2D transfer function can map RGBA values separately to every combination of density data and gradient magnitude

2D Transfer function and its parameters (Wan et al. 2009)
2D Transfer Functions

- Example: Rectangular 2D transfer function editor

Images by Gordon Kindleman and Joe Kniss
Lecture Overview

- Volume Rendering
  - Overview
  - Transfer Functions
  - Rendering
Volume Rendering Techniques

- Iso-surface
- Cross-sections
- Direct volume rendering (DVR)
  - Slicing with 2D textures
  - Translucent textures with image plane-aligned 3D textures
  - MIP
- Spatial constraints
  - Region of interest (cubic, spherical, slab)
  - Clipping plane
Volume Rendering Techniques

Maximum Intensity Projection

Iso-Surface

Transparent Iso-Surfaces

Raycasting (DVR)

Raycasting and Iso-Surface

Volume Rendering Outline

Data Set → 3D Rendering → Classification

Rendering done in real-time on commodity graphics hardware
Ray Casting

- Software Solution

- **Image Plane**
- **Data Set**

- **Numerical Integration**
- **Resampling**

- **High Computational Load**
Ray Casting

- Software Solution

Image Plane

Eye

Data Set

- Numerical Integration
- Resampling

High Computational Load
Plane Compositing

Proxy geometry (Polygonal Slices)
Compositing

- **Maximum Intensity Projection**
  - No emission/absorption
  - Simply compute maximum value along a ray

![Emission/Absorption](A) ![Maximum Intensity Projection](B)

*Emission/Absorption*  *Maximum Intensity Projection*
2D Textures

- Draw the volume as a stack of 2D textures

*Bilinear Interpolation in Hardware*

- Decomposition into axis-aligned slices

- 3 copies of the data set in memory
2D Textures: Drawbacks

- Sampling rate is inconsistent

- Emission/absorption slightly incorrect

- **Super-sampling on-the-fly impossible**
3D Textures

For each fragment:
interpolate the texture coordinates *(barycentric)*

Texture-Lookup:
interpolate the texture color *(trilinear)*
3D Textures

**3D Texture**: Volumetric Texture Object
- Trilinear Interpolation in Hardware
- Slices parallel to the image plane

- One large texture block in memory
Comparison of 2D with 3D Texturing

Left: 2D textures, right: 3D textures
[Lewiner2006]
Resampling via 3D Textures

- *Sampling rate is constant*

- Supersampling by increasing the number of slices
Shadows

Volume rendering with shadows
(from GPU Gems)
// init the 3D texture
glEnable(GL_TEXTURE_3D_EXT);

GenTextures(1, &tex_glid);

BindTexture(GL_TEXTURE_3D_EXT, tex_glid);

// texture environment setup
TexParameteri( GL_TEXTURE_3D_EXT, GL_TEXTURE_MIN_FILTER, GL_LINEAR );

TexParameteri( GL_TEXTURE_3D_EXT, GL_TEXTURE_MAG_FILTER, GL_LINEAR );

TexParameteri( GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE );

TexParameteri( GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );

TexParameteri( GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE );

// load the texture image
TexImage3DEXT(GL_TEXTURE_3D_EXT, // target
0, // level
GL_RGBA, // color storage
(int) tex_ni(), // width
(int) tex_nj(), // height
(int) tex_nk(), // depth
0, // border
GL_COLOR_INDEX, // format
GL_FLOAT, // type
_texture ); // allocated texture buffer

PixelTransferi(GL_MAP_COLOR, GL_FALSE);
Demo: DeskVox

- DeskVox was created at IVL/Calit2
Videos

- Human head, rendered with 3D texture:
  - http://www.youtube.com/watch?v=94_Zs_6AmQw
- GigaVoxels:
  - http://www.youtube.com/watch?v=HScYuRhgEJw
References

- **Volume rendering tutorial with source code**

- **Simian volume rendering software**
Lecture Overview

- Deferred Rendering Techniques
  - Deferred Shading
  - Screen Space Ambient Occlusion
  - Bloom
  - Glow
Deferred Rendering

- Opposite to Forward Rendering, which is the way we have rendered with OpenGL so far
- Deferred rendering describes post-processing algorithms
  - Requires two-pass rendering
  - First pass:
    - Scene is rendered as usual by projecting 3D primitives to 2D screen space.
    - Additionally, an off-screen buffer (G-buffer) is populated with additional information about the geometry elements at every pixel
      - Examples: normals, diffuse shading color, position, texture coordinates
  - Second pass:
    - An algorithm, typically implemented as a shader, processes the G-buffer to generate the final image in the back buffer
Lecture Overview

- Deferred Rendering Techniques
  - Deferred Shading
  - Screen Space Ambient Occlusion
  - Bloom
  - Glow
- The Future of Computer Graphics
Deferred Shading

- Postpones shading calculations for a fragment until its visibility is completely determined
  - Only fragments that really contribute to the image are shaded

- Algorithm:
  - Fill a set of buffers with common data, such as diffuse texture, normals, material properties
  - For the lighting just render the light extents and fetch data from these buffers for the lighting computation

- Advantages:
  - Decouples lighting from geometry
  - Several lights can be applied with a single draw call: more than 1000 light sources can be rendered at 60 fps

- Disadvantages:
  - Consumes more memory, bandwidth and shader instructions than traditional rendering
Reference

- Deferred Shading Tutorial:
Lecture Overview

- Deferred Rendering Techniques
  - Deferred Shading
  - Screen Space Ambient Occlusion
  - Bloom
  - Glow
- The Future of Computer Graphics
Screen Space Ambient Occlusion

- Screen Space Ambient Occlusion is abbreviated as SSAO
- “Screen Space” refers to this being a deferred rendering approach
- Rendering technique for approximating ambient occlusion in real time
- Developed by Vladimir Kajalin while working at Crytek
- First use in 2007 PC game Crysis
Ambient Occlusion

- Attempts to approximate global illumination
  - Very crude approximation
- Unlike local methods like Phong shading, ambient occlusion is a global method
  - Illumination at each point is a function of other geometry in the scene
- Appearance achieved by ambient occlusion is similar to the way an object appears on an overcast day
  - Example: arm pit is hit by a lot less light than top of head
- In the industry, ambient occlusion is often referred to as "sky light"
SSAO Demo

- Screen Space Ambient Occlusion (SSAO) in Crysis
  - http://www.youtube.com/watch?v=ifdAILHTcZk
Basic SSAO Algorithm

- **First pass:**
  - Render scene normally and write z values to g-buffer’s alpha channel

- **Second pass:**
  - Pixel shader samples depth values around the processed fragment and computes amount of occlusion, stores result in red channel
  - Occlusion depends on depth difference between sampled fragment and currently processed fragment

Ambient occlusion values in red color channel

*Source: www.gamerendering.com*
SSAO With Normals

- **First pass:**
  - Render scene normally and copy z values to g-buffer’s alpha channel and scene normals to g-buffer’s RGB channels

- **Second pass:**
  - Use normals and z-values to compute occlusion between current pixel and several samples around that pixel

---

![No SSAO](image1)

![With SSAO](image2)
SSAO Discussion

**Advantages:**
- Deferred rendering algorithm: independent of scene complexity
- No pre-processing, no memory allocation in RAM
- Works with dynamic scenes
- Works in the same way for every pixel
- No CPU usage: executed completely on GPU

**Disadvantages:**
- Local and view-dependent (dependent on adjacent texel depths)
- Hard to correctly smooth/blur out noise without interfering with depth discontinuities, such as object edges, which should not be smoothed out
References

- Nvidia’s documentation:

- SSAO shader code from Crysis:

- Another implementation:
Lecture Overview

- Deferred Rendering Techniques
  - Deferred Shading
  - Screen Space Ambient Occlusion
  - Bloom
  - Glow
- The Future of Computer Graphics
Bloom Effect

Bloom gives a scene a look of bright lighting and overexposure.

Left: no bloom, right: bloom.
Source: http://jmonkeyengine.org
Bloom Shader

- Post-processing filter: applied after scene is rendered normally
- Step 1: Extract all highlights of the rendered scene, superimpose them and make them more intense
  - Operates on back buffer
  - Often done with off-screen buffer smaller than frame buffer
  - Highlights found by thresholding luminance
- Step 2: Blur off-screen buffer, e.g., with Gaussian blurring
- Step 3: Composite off-screen buffer with back buffer

Bloom shader render steps. Source: http://www.klopfenstein.net
References

- Bloom Shader

- GLSL Shader for Gaussian Blur
  - http://www.ozone3d.net/tutorials/image_filtering_p2.php
Lecture Overview

- Deferred Rendering Techniques
  - Deferred Shading
  - Screen Space Ambient Occlusion
  - Bloom
  - Glow
- The Future of Computer Graphics
Glow Effects

- Glows and halos of light appear everywhere in the world
- They provide powerful visual cues about brightness and atmosphere
- In computer graphics, the intensity of light reaching the eye is limited, so the only way to distinguish intense sources of light is by their surrounding glow and halos
- In everyday life, glows and halos are caused by light scattering in the atmosphere or within our eyes

A cityscape with and without glow. Source: GPU Gems
Glow vs. Bloom

- Bloom filter looks for highlights automatically, based on a threshold value
- If you want to have more control over what glows and does not glow, a glow filter is needed
- Glow filter adds an additional step to Bloom filter: instead of thresholding, only the glowing objects are rendered
- Render passes:
  - Render entire scene back buffer
  - Render only glowing objects to a smaller off-screen glow buffer
  - Apply a bloom pixel shader to glow buffer
  - Compose back buffer and glow buffer together
References

- GPU Gems Chapter on Glow

- Bloom and Glow
The Future of Computer Graphics

- ACM SIGGRAPH Asia, 19.11.-22.11.2014 in Hong Kong (3:18)
  - [http://www.youtube.com/watch?v=FUGVF_eMeo4](http://www.youtube.com/watch?v=FUGVF_eMeo4)
- ACM SIGGRAPH, August 10-14, 2014, Vancouver
  - [Student volunteer application](http://www.youtube.com/watch?v=aseq4T81P7g) deadline: Feb 9, 2014

- Cryengine 4 Trailer
  - [http://www.youtube.com/watch?v=aseq4T81P7g](http://www.youtube.com/watch?v=aseq4T81P7g)
Good luck with your final projects!